Theor Chim Acta (1993) 84: 413–421

Theoretica Chimica Acta

Where are embarrassingly parallel problems? The atom-diatom quasiclassical reactivity

Antonio Laganà[1], Osvaldo Gervasi[2], Ranieri Baragli[3], Domenico Laforenza[3], and Raffaele Perego[3]

[1] Dipartimento di Chimica, Università di Perugia, I-06100 Perugia, Italy

[2] Centro di Calcolo, Università di Perugia, Perugia, Italy

[3] CNUCE, Pisa, Italy

Received October 1, 1991/Accepted February 7, 1992

Summary. The apparently embarrassingly parallel problem of calculating a batch of quasiclassical trajectories to evaluate cross sections or rate constants has been analyzed. The study shows that only an accurate evaluation of the performance parameters and a consequent appropriate restructuring of the computer code allows the achievement of high speedup and efficiency especially when large calculations are implemented on massively parallel systems.

Key words: Quasiclassical reactivity – Atom-diatom – Reactive cross sections – Parallel computing

1. Introduction

At the Meeting "Parallel Computing for Chemical Reactivity" held in Perugia (Italy) in August 1990 several approaches to the calculation of reactive scattering properties were described as *embarrassingly parallel* [1, 2]. This is due to the fact that scattering properties are, in general, expressed as a sum over the appropriate indexes of the square modulus of the detailed scattering matrix elements. These are, usually, obtained via a time-consuming integration of different subsets of differential equations resulting into an ensemble of naturally separable (substantially uncoupled) computational processes. However, though appearing as an embarrassingly parallel problem, scattering calculations cannot be efficiently parallelized if a detailed evaluation of the parallel performance parameters and an appropriate restructuring of the computer code is not carried out.

As discussed at the "Parallel Computing in Chemical Physics" meeting held in Willowbrook (Illinois, USA) in July 1991, this is the case of classical trajectory calculations. Significant difficulties have been encountered when parallelizing Molecular Dynamics codes dealing with large molecular systems [3, 4]. More successful are quasiclassical calculations of reactive properties of small systems [5]. To achieve such a result, however, we have carried out a detailed analysis of the computational bottle-necks and a careful exploitation of the potentially parallel features of the trajectory method. Goal of the paper is to report the details of such an investigation.

To this end, after revisiting the main features of the quasiclassical trajectory approach (Sect. 2) and the structure of the corresponding computer code (Sect.

3), we introduce and discuss a few main performance parameters (Sect. 4). The appropriate restructuring of the code based on the evaluation of these performance parameters is discussed in Sect. 5. The final performance evaluation of the restructured code is reported in Sect. 6.

2. The quasiclassical approach to atom-diatom reactivity

Though extensively described in the literature [6, 7], the main features of the quasiclassical approach to atom-diatom reactivity are again outlined in this section to facilitate the understanding of the potential parallelism of the code. A quasiclassical approach starts from the assumption that molecular motions can be treated by considering the nuclei as mass points obeying to the classical mechanics and that the discrete nature of internal energy can be regained both at the beginning and at the end of the calculation by merely imposing initial values to correspond to reactant vibrotational eigenvalues and boxing final energies. For a given value of the translational energy E_{tr}, the detailed cross section from the reactant vibrotational vj to the product vibrotational $v'j'$ state is defined as:

$$S_R^{vj,v'j'}(E_{tr}) = C \int_0^1 d\xi_1 \int_0^1 d\xi_2 \cdots \int_0^1 d\xi_k f_{vj,v'j'}(E_{tr}, \xi_1, \xi_2, \ldots, \xi_k) \qquad (1)$$

with C being a normalization constant and k being 5 for atom-diatom systems when a proper choice of the reference frame orientation is made. In this case, the ξ variables are defined as:

$$\begin{aligned} \xi_1 &= (1 - \cos \vartheta^0)/2 \\ \xi_2 &= \eta^0/(2\pi) \\ \xi_3 &= (b/b_{\max})^2 \\ \xi_4 &= \beta^0/\pi \\ \xi_5 &= \phi^0/(2\pi) \end{aligned} \qquad (2)$$

with the superscript "^{0}" meaning the initial value. In the chosen reference frame the three atoms lie on the zx plane, the angle ϑ gives the orientation of the diatom with respect to the z axis connecting the atom A to the diatom BC, the phase angle η determines the diatomic oscillator elongation while the initial atom-diatom distance is chosen large enough to render the related interaction negligible. These quantities fully determine the two position vectors $\boldsymbol{W}_A$ and $\boldsymbol{W}_C$ (the third one $\boldsymbol{W}_B$ is uniquely determined by the choice of locating the axes origin at the center of mass of the triatom). Initial values of moments are derived from the atom-diatom relative velocity $v_{A,BC}$ and from the characteristics of the diatomic rotation and vibration. To this end, the relative atom-diatom velocity is expressed in terms of the initial collision energy E_{tr}, the impact parameter b (defined as the distance of the diatom center of mass from $v_{A,BC}$ whose maximum value $b_{\max}$ is given by the largest impact parameter at which reaction still occurs) and the angle β formed by the projection of $v_{A,BC}$ onto the xy plane and the y axis. The other values to be supplied are the vibrational and rotational energy (E_v and E_j respectively) as well as the angle ϕ formed by the rotational angular momentum of the diatom with the yz plane.

Once initial values of positions *W* and momenta *P* are worked out, the time evolution of the system is calculated by integrating numerically the equations:

$$\begin{aligned}\frac{dW_{Is}}{dt} &= \frac{\partial H^C}{\partial P_{Is}}\\ \frac{dP_{Is}}{dt} &= -\frac{\partial H^C}{\partial W_{Is}}\end{aligned} \tag{3}$$

linking the time derivatives of positions and momenta to the partial derivatives of the classical Hamiltonian [8] (H^C):

$$H^C = \sum_I \frac{P_I^2(W)}{2M_I} + V(W) \tag{4}$$

and I being either A or C and s indicating any of the three cartesian axes. The curve connecting calculated position values is the trajectory followed by the system during its evolution. Usually, the conservation of the total energy E and total angular momentum $\boldsymbol{J}$ are not incorporated into the motion equations to further reduce their number. Therefore, the conservation of these quantities can be used as an *a posteriori* check of the accuracy of the numerical integration.

By assigning a given final classical state $v'_c j'_c$ to the closest quantum state $v'j'$, the detailed reactive cross section can be written as:

$$S_R^{vj,v'j'}(E_{tr}) = \pi b_{\max}^2 P_{vj,v'j'}(E_{tr}) = \pi b_{\max}^2 \frac{N_R^{vj,v'j'}}{N} \tag{5}$$

where N is the number of integrated trajectories and $N_R^{vj,v'j'}$ is the number of trajectories that starting from reactants in the vj state can be assigned to the final state $v'j'$. Quite recently the detail of experimental measurements has led to the determination of properties of oriented and aligned target molecules [9]. In this case, b, ϕ^0, β^0 and ϑ^0 are not independent variables being linked by relationships ensuring the wanted orientation or alignment of the considered vectors.

More frequently, however, experimental quantities are less detailed than the state-to-state differential cross section and, therefore, a further averaging of the calculated probability is required. In many gas kinetics studies, for example, what is actually needed is the temperature (T) and vibrational state dependent reactive rate constant $k_{v,v'}(T)$ that is given by the following expression:

$$\begin{aligned} k_{v,v'}(T) &= \left(\frac{2}{kT}\right)^{3/2} \frac{e^{-E_v/kT}}{(\pi\mu_{A,BC})^{1/2} Q_{VR}(T)} \sum_{j=0}^{\infty} (2j+1)\, e^{-E_j/kT}\\ &\times \int_0^\infty dE_{tr} E^{tr}\, e^{-E_{tr}/kT} S_R^{vj,v'}(E_{tr}) \end{aligned} \tag{6}$$

where k is Boltzmann's constant, $\mu_{A,BC}$ is the $A+BC$ reduced mass, $S_R^{vj,v'}(E_{tr})$ is the degeneracy averaged detailed reaction cross section summed over final AB rotational states and $Q_{VR}(T)$ is the BC vibrotational partition function.

3. The structure of the computer code

On conventional computers, the code can be organized in three sections. The first section is dedicated to input data and to calculate variables of common use throughout the code. This section exploits the different options for selecting the energy distribution among reactants and works out the mass factors, convergence and accuracy check variables, debugging flags and limits of the integration domain.

The second section iterates over the number of trajectories to be integrated the necessary operations. The first operation of this section is concerned with the generation of a string of pseudo-random numbers and its conversion into validated initial values of the integration variables. The generation of the initial parameters has to be strictly sequential to guarantee the reproducibility of the results even when working conditions (e.g. the integration stepsize) vary. To this end, the pseudo-random sequence is generated using a routine that from a given seed generates a new seed and a random number in the interval 0–1. The second operation of this section consists of integrating the twelve Hamilton first order differential equations using a predictor corrector method started by the needed number of Euler steps. The integration is carried out to a point where one of the three internuclear distances is again large enough to consider the process ended. The third operation of this section consists of working out the product characteristics from the final value of the $\boldsymbol{W}$ and $\boldsymbol{P}$ vectors. To this end, the motion along the diatomic internuclear distance is separated out to recover the vibrational and the rotational energy of the product molecule as well as its rotational angular momentum, vibrational phase and orientation. This allows an update of the statistical indicators related to energy and vector product distributions.

The third and last section of the code takes care of the final print-out of the calculated distributions and of the evaluation and print-out of the reactive cross section.

4. Performance parameters for parallel executions

The main goal for parallelizing a code is the achievement of a high computing speed. To achieve a high speed the computing time has to be maximized while reducing all waiting or communicating times. To this end, it is worth defining here the following performance parameters: the speedup (S), the scaled speedup (S_S) and the efficiency (E).

The usual speedup S is defined as the ratio between the time T_S needed to run the application on a single processor and the time T_P needed to run the application in parallel on the wanted number of processors:

$$S = \frac{T_S}{T_P} \tag{7}$$

Obviously such a quantity does increase when T_P becomes smaller. However, though S has a simple definition, its actual evaluation may become unpractical. As an example, for large programs T_S may be difficult to evaluate because they are extremely large. It is also possible that the full program (including data) does not fit into the node memory of a (distributed memory) parallel machine or that the implementation of the program on a many processor machine (this is particularly true for massively parallel distributed memory environments) implies such a deep restructuring to make the two versions of the code completely different.

A more appropriate way of quantifying the gain in speed is to make use of the so-called scaled speedup S_S defined as [10]:

$$S_S = \frac{s + p \cdot P}{s + p} \tag{8}$$

where s is the time spent for running the sequential part of the code while p is the time spent to run the parallel part of the code and P the number of processors. The scaled speedup is measured by varying the number of nodes used while keeping constant the number of processes assigned to individual nodes. As a result, the scaled speedup is a measure of how the code scales when the number of processes and the number of processors are proportionally increased (i.e. is a measure of how close the execution time of a problem of size d on P processors is related to that of size kd on kP processors).

The efficiency E is usually defined as:

$$E = \frac{S}{P} \tag{9}$$

and is a measure of to what extent a processor is engaged in useful work (maximum efficiency is 1).

The optimization of the speedup and efficiency when implementing an application on a parallel architecture is mainly a matter of distribution of the work among the processors. To do this, one needs to further define the two quantities T_R and T_W (the sum of the individual node running (t_{r_i}) and waiting (t_{w_i}) times, respectively). Waiting times are those spent in activities other than computing (e.g. synchronization, communication, etc.). By expressing the time p spent to run the parallel part of the code in terms of the individual node times:

$$p = t_{r_i} + t_{w_i} = t_{r_j} + t_{w_j} \tag{10}$$

the time T_P spent by the code for running on a parallel machine can be expressed as:

$$T_P = s + p = s + \sum_i^P \frac{t_{r_i} + t_{w_i}}{P} = s + \frac{T_R}{P} + \frac{T_W}{P} \tag{11}$$

establishing a relationship between the speedup and the total running T_R and waiting T_W times of the P processors.

5. The parallel restructuring of the trajectory program

As evident from the discussion of Sect. 3, the operation (or equivalently the part of the trajectory code) that can be easily distributed among several processors for parallel execution with little restructuring effort is the integration of the motion equations. Once defined, in fact, the six initial position values and the corresponding initial momenta (the generation of these values, as already mentioned, has to be strictly sequential) the integration of the motion equations of each individual trajectory is a fully independent process. Therefore, by adopting a farm parallel model [11, 12], this operation could be distributed among the worker nodes provided that for each trajectory the master takes care of determining the initial values of the variables and distributing them to the workers. However, such a simplistic organization of the code tailored on the embarrassingly parallel structure of the trajectory approach, fails to guarantee high speedups. This is especially true when, as is frequently the case of scattering calculations, a large number of trajectories is needed (N is typically of the order of several thousands) and use has to be made of massively parallel architectures.

The time T_P needed for a parallel execution is, in fact, directly proportional to both s and T_W. Why an increase of s makes the computing time associated with a simplistic parallel organization of the trajectory code dramatically large can be easily understood. At the beginning of the trajectory loop, in fact, the initial conditions need to be generated and validated. This amounts to a nonnegligible fraction of the scalar time s. The fact that this calculation has to be repeated every time a trajectory is started means that its actual contribution to the global s value is N times that of a single trajectory. On top of that, the amount of data to be sent to the worker processors to start the integration of the assigned trajectory, though small, is not negligible. This contributes to make individual communication times (and as a consequence T_W) large especially when the number of nodes to be served becomes high (with a consequent heavy contention of the network). Both arguments apply also to the collection of the results once the trajectory integration comes to an end. Actually, the amount of data sent back by a trajectory for updating the statistical analysis is much larger than that of initial values being concerned with several scalar, vector and matrix variables.

For these reasons, we restructured the code by suppressing or minimizing as much as possible all the communications. For the initial conditions the choice was made of incorporating into the section of the code allocated to the worker nodes their generation and validation. Only the generation of the first seed of each trajectory was kept at master level. Once generated, the seed vector (of dimension N) is distributed to the worker processors. This has the advantage of relieving the master from generating and validating initial conditions and limits the transmission between the master and the worker to a single integer number (the sequential number of the trajectory to be integrated) while preserving the strict sequentiality of the initial conditions generation. As a result, the sequential work is dramatically reduced and turned into a parallel one.

As far as the results' return is concerned, we have been able to suppress the return of individual trajectory results by performing a local (in-node) update of the statistical indexes. Statistical indexes are collected only at the end of the calculation of all trajectories to perform the global statistical analysis. A further reduction of the transmission time has been obtained on the hypercube by adopting for the data collection a dimensional collapsing algorithm [13].

As a result of the above restructuring, the communication times of the different nodes became very similar. The differences (as shown by Fig. 1) amount

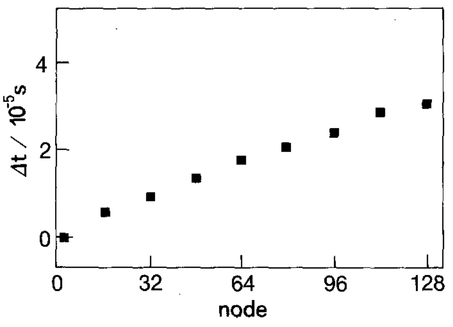

Fig. 1. The communication time increment (Δt) plotted as a function of the node identity number for a run of 4096 trajectories on a 128 node Ncube 2

to a few μs seconds for a run of 4096 trajectories on a Ncube 2 consisting of 128 nodes (on the average 32 trajectories per node) negligibly affecting the global computing time (communication time is only 0.002% of the total parallel time p being that for node 1 of 1.0000665 s).

6. Performance evaluation

After the program was restructured as discussed in the previous section, the waiting time reduced basically to that needed for the dispatch of local statistical indexes to the master processor. This made relevant the evaluation of the parallel performance of the trajectory code as well as the extension of the calculations to large batches of trajectories and highly parallel architectures. As already mentioned in Sect. 4, in these cases the scaled speedup is an appropriate performance index. The scaled speedup calculated on an nCUBE/10 (a machine of the first nCUBE generation) by running 16 trajectories per node (a total of 8192 trajectories on 512 nodes) is shown in Fig. 2. In this case, a scaled speedup of 15.95 for 16 processors and of 508.4 for 512 processors was reached. The linearity shown by the logarithm of the scaled speedup when plotted as a function of the number of processors (on a logarithmic scale) demonstrates the perfect scalability of the restructured trajectory code.

The importance of using the scaled speedup is evidenced by the measurement of the relative importance of the waiting time when running different batches of trajectories per node. An example of the percentage of T_P wasted as T_W when running different numbers of trajectories (64, 128, 256 and 512 respectively) on a fixed number of nodes (16 in the case of the figure) is shown in Fig. 3. As clearly shown by the figure the percentage of T_W decreases when the number of trajectories allocated to each node increases. This effect becomes increasingly larger when the dynamical allocation (in the adopted farm model a new trajectory is assigned to a node as soon as it is disengaged from calculating the previous trajectory) is constrained by an increase of the granularity. In fact, the dispatching of a subset of trajectories rather than a single one (granularity 1) reduces the number of processes per node while making each process larger and the node load less likely to be balanced. As a result, in spite of the fact that an increase of the granularity reduces the communication frequency, the elapsed time increases. This is already apparent in Fig. 3 where a more dynamical assignment of the trajectories leads to a lower percentage of waiting time. A

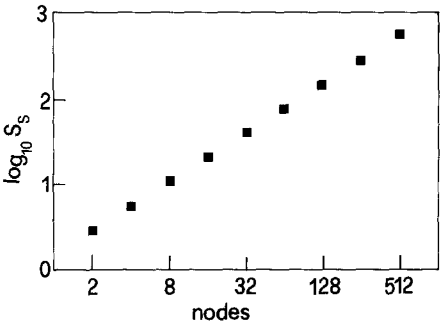

Fig. 2. The logarithm of the scaled speedup S_S plotted as a function of number of the nCUBE/10 nodes used

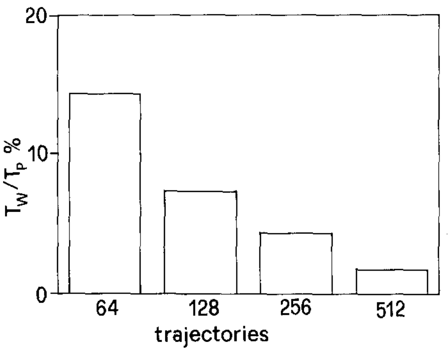

Fig. 3. The fraction of T_P spent as waiting time T_W plotted as a function of the number of trajectories run on 16 nodes

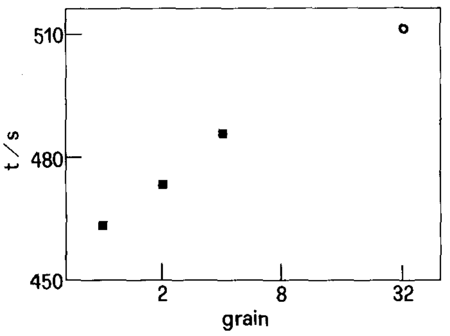

Fig. 4. Elapsed time measured for calculating 4086 trajectories on a 128 node Ncube 2 plotted as a function of the granularity (*squares*). The result of implementing a SPMD parallel version of the code is also reported as a *circle* at the location of the equivalent granularity farm

more direct evidence of the effect of increasing the granularity of the calculation is shown in Fig. 4 where elapsed times for granularity 1, 2 and 4 are shown. In the figure, the computing time needed for integrating a fixed amount of 4096 trajectories on 128 nodes is plotted (squares) as a function of the granularity adopted. The figure clearly shows that the computing time increases linearly with the granularity.

Further evidence for this can be obtained from elapsed times measured for the implementation of the code based on the SPMD model [14]. In the SPMD model, the whole sequential program is copied into all nodes with minor modifications. This allows the node to open the input file, generate the appropriate subset of initial conditions and integrate the related motion equations. Once the work is completed, a library routine performs a global adding of the statistical indexes by mapping an addition tree onto the hypercube. In our case, this means the running of 32 trajectories per node. For this reason, our SPMD results can be compared with those of a farm model at granularity 32. As can be easily seen from Fig. 4 where the SPMD result is reported as a circle in the location corresponding to that of the farm having granularity 32, the performance of a SPMD parallel implementation is substantially aligned with the farm ones.

7. Conclusions

We have restructured for parallel execution a quasiclassical trajectory program. In principle, this application should fall in the class of the so-called embarrassingly parallel codes. However, when large batches of trajectories are to be run and use has to be made of highly parallel machines, very little speedup is gained when adopting a simplistic parallel implementation based on the separation of the trajectory integration and the dispatch of this program section to the worker processors. To optimize the restructuring of the trajectory code, we have calculated some efficiency parameters of the application to find out that computational bottle-necks to the parallelism can be eliminated by minimizing the node waiting time.

To this end, we have restructured the application to minimize node communications by reducing the initial conditions and suppressing the collection of statistical individual trajectory indexes. Results are returned only at the end of the calculation. Only after such a restructuring, we have been able to record impressive speedups using a farm model parallel organization. The analysis of the performance parameters has also led to the conclusion that smaller granularities guarantee higher speedups. In fact, in spite of the lower message passing associated with higher granularities, the resulting larger load unbalance worsens the performance. This has also been confirmed by the implementation of the SPMD model that approximately performed (in our case) as the farm case having an equivalent granularity.

Acknowledgments. Partial financial support from CNR (Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo) is acknowledged.

References

1. (1990) Proceedings of the Meeting Parallel Computing for Chemical Reactivity, Perugia, Italy
2. (1991) Theor Chim Acta 79
3. (1991) Proceedings of the Meeting on Parallel Computing for Chemical Physics, Willowbrook, USA
4. Theor Chim Acta (this issue)
5. Laganà A, Garcia E, Gervasi O, Baraglia R, Laforenza D, Perego R (1991) Theor Chim Acta 79:323
6. Bunker DL (1971) Methods Comput Phys 10:287
7. Clementi E (1985) J Phys Chem 89:4426
8. Marion JB (1965) Classical dynamics of particles and systems. Academic Press, NY
9. (1987) J Phys Chem 91
10. Gustafson JL, Montry GR, Benner RE (1988) SIAM J Scient Stat Comput 9(4):609
11. Helliot RJ, Hoare CAR (1989) Scientific applications on multiprocessors. Prentice Hall, NY
12. Protchard DJ, Askew CR, Carpenter DB, Glendinning I, Hey AJG, Nicole DA (1987) Practical parallelism using transputer array. In: Lecture Notes in Computer Science. Springer-Verlag, Berlin, p 258
13. Baraglia R, Ferrini R, Laforenza D, Perego R, Laganà A, Gervasi O (submitted) J Math Chem
14. Baraglia R, Ferrini R, Laforenza D, Perego R, Gervasi O, Laganà A (1991) High Performance Computing 2. In: Durand M, El Dabaghi F (eds) North-Holland, Amsterdam, 1991, p 17